(2)

Systems
Optimization
Laboratory

# A Vectorization Algorithm for the Solution of Large, Sparse Triangular Systems of Equations

by
Samuel K. Eldersveld and Martin C. Rinard

TECHNICAL REPORT SOL 90-1

January 1990

Department of Operations Research
Stanford University
Stanford, CA 94305

4

SYSTEMS OPTIMIZATION LABORATORY
DEPARTMENT OF OPERATIONS RESEARCH
STANFORD UNIVERSITY
STANFORD, CALIFORNIA 94305-4022

# A Vectorization Algorithm for the Solution of Large, Sparse Triangular Systems of Equations

by
Samuel K. Eldersveld and Martin C. Rinard

TECHNICAL REPORT SOL 90-1

January 1990

Research and reproduction of this report were partially supported by the National Science Foundation grant ECS-8715153, and the Office of Naval Research Grant N00014-90-J-1242.

# A VECTORIZATION ALGORITHM FOR THE SOLUTION OF LARGE, SPARSE TRIANGULAR SYSTEMS OF EQUATIONS

Samuel K. ELDERSVELD§* and Martin C. RINARD‡

§Systems Optimization Laboratory, Department of Operations Research
Stanford University, Stanford, California 94305–4022, USA
‡Computer Science Department
Stanford University, Stanford, California 94305–2140, USA

Technical Report SOL 90-1†
January 1990

**Abstract**

A new method is given for use with vector computers on applications that require multiple solutions with identically patterned triangular factors and different right-hand sides. A key feature is that a vectorization algorithm is used to place the nonzeros from the factors in a few long vectors. The method is shown to work well when incorporated into the mathematical programming system MINOS and tested on 30 linear programming test problems.

## 1. Introduction

Many application problems require multiple solutions with large, sparse triangular systems of equations. Often these systems are identical, or share the same nonzero pattern. The triangular system usually occurs after an arbitrary (not necessarily square) matrix has been factorized into two triangular matrices: $B = LU$.

This paper presents a general method for vectorizing any application that requires multiple solves with a sparse $L$ or $U$. The experimental results demonstrating the utility of our vectorization method come from the simplex method for linear programming. In the simplex method, the triangular systems arise from the factorization of a nonsymmetric basis matrix $B$. During each iteration $k$ of the simplex method, the basis $B_k$ is used to solve for the search direction $p$ and the dual variables $\pi$ in the following linear systems:

$$B_k p = a_q \quad \text{and} \quad B_k^T \pi = c_k.$$

*S.K. Eldersveld's research was supported by an IBM Graduate Technical Fellowship.

†The material contained in this report is based upon research supported by the National Science Foundation Grant ECS-8715153 and the Office of Naval Research Grant N00014-90-J-1242.

```
$y \leftarrow b$
for $j = 1$ to $n$ do
    $y_j \leftarrow y_j / l_{jj}$
    for all off-diagonal nonzeros $l_{ij}$ in column $j$ do
        $y_i \leftarrow y_i - l_{ij} \times y_j$
```

Figure 1: Lower triangular solve.

An update is then applied in which a single column of $B_k$ is replaced. Using *Bartels-Golub* [Bar71] or *Forrest-Tomlin* [FT72] updates, the basis at iteration $k$ may be represented as

$$B_k = L_k U_k = (L_0 M_k) U_k, \tag{1.1}$$

where $L_0$ is the result of a direct factorization of $B_0$, $M_k$ is a product of updates, and $U_k$ is updated explicitly. The factor $L_0$ may be used for 100 or more iterations before a "fresh" factorization is performed. For a full discussion of the process of factorization and updating, the reader is referred to [GMSW87]. With either a *product-form* update or a *block-LU* update, both factors $L_0$ and $U_0$ of the initial basis $B_0$ may be used for many iterations. The block-$LU$ update and its efficiency for vector computers is discussed in [ES90].

For problems in which the data matrix is very sparse, the factors themselves are often very sparse. This is particularly the case with large-scale linear programming where each column of the factors may have very few nonzeros.

The next section presents a naive vectorization method for solves with these factors, while Section 3 presents an optimal vectorization algorithm for such solves. Computational results for the optimal vectorization method are given in Section 4.

## 2. Solutions of systems involving $B$, $L$ and $U$

We assume an available factorization $B = LU$. Under this assumption, a standard method for solving a set of linear equations $Bx = b$ is to use forward and back substitution to solve $Ly = b$ and $Ux = y$. We shall focus on the system $Ly = b$ for the remainder of the paper.

When the $n \times n$ factor $L$ is stored by columns, it is convenient to use the forward substitution algorithm given in Figure 1. Note that every off-diagonal nonzero of $l_{ij}$ generates the operation $y_i \leftarrow y_i - l_{ij} \times y_j$; we call this $l_{ij}$'s operation.

When $L$ is dense, both the multiply and subtract in line 5 of the algorithm can become vector operations. Because the computation associated with a column $L_j$ depends on the computation associated with column $L_{j-1}$, each column must generate a separate vector operation.

Sparse matrices are often stored by column with zero elements eliminated; an associated index array gives the row index for each nonzero. On machines with hardware gather/scatter operations, the obvious way to vectorize a solve with $L$ is again to make each column's computation a separate vector operation. To perform the computation associated with column $L_j$, the algorithm would broadcast the element $y_j$ into adjacent locations of a work vector (one per subdiagonal nonzero $l_{ij}$), perform the vector multiply on this work vector, gather the elements $y_i$ from which the products must be subtracted into adjacent locations of another work vector, subtract the two work vectors, and scatter the elements $y_i$ back into the appropriate locations in $y$.

The problem with this vectorization is that for many sparse matrices, the resulting vectors are so short that very little speedup is observed. It is often possible to increase the average vector length by taking advantage of the sparsity pattern of the matrix to schedule operations from different columns of $L$ into the same vector operation.

## 3. The Vectorization Algorithm

For the purposes of exposition, we will assume that $L$ is unit-diagonal. This constraint will be removed later. Given $L$, our vectorization algorithm must generate a sequence of vectors. Each vector is a sequence of scalar operations of the form $y_i \leftarrow y_i - l_{ij} \times y_j$; we will call such a sequence of vectors a vector schedule. A vector schedule solves $Ly = b$ if and only if it satisfies the following constraints:

1. Every scalar operation in the vector schedule is the operation of some off-diagonal nonzero of $L$.
2. Every off-diagonal nonzero's operation appears exactly once in the vector schedule.
3. Every vector contains at most one operation from row $i$ of $L$ ($i = 1, \ldots, n$).
4. All vectors containing an operation from row $i$ appear before any vector containing operations from column $i$.

These constraints suggest the algorithm given in Figure 2. This algorithm maintains $n$ queues of nonzeros, one for each row of $L$. $Q_i$ contains all nonzeros from row $i$ of $L$ that are ready to be scheduled. The algorithm does not enqueue a nonzero $l_{ij}$ into queue $Q_i (i > j)$, until it has scheduled all operations from row $j$, so the generated schedule satisfies constraint 4.

The algorithm schedules each vector by scheduling one operation from each nonempty queue. Because each nonzero $l_{ij}$ appears only in queue $Q_i$, the generated schedule satisfies constraint 3.

If there are no off-diagonal nonzeros in row $j$, the algorithm enqueues the off-diagonal nonzero $l_{ij}$ before scheduling any vectors. Because the algorithm runs until all queues are empty, $l_{ij}$ will eventually be dequeued and its operation scheduled. Because there are no off-diagonal nonzeros in row $j$, the algorithm will not enqueue $l_{ij}$ again.

If there are off-diagonal nonzeros in row $j$, the algorithm enqueues nonzeros $l_{ij}$ in column $j$ only when the last nonzero from row $j$ is dequeued. If each nonzero in row $j$ is dequeued exactly once, then the last nonzero in row $j$ will be dequeued exactly

```
for $i = 1$ to $n$ do
    $Q_i \leftarrow$ empty queue
for each row $j$ with no off-diagonal nonzeros do
    for all off-diagonal nonzeros $l_{ij}$ in column $j$ do
        enqueue $l_{ij}$ in queue $Q_i$
$v \leftarrow 1$
while some queue is nonempty do
    SCHED$\leftarrow \phi$
    for each nonempty $Q_i$ do
        dequeue a nonzero $l_{ij}$ from queue $Q_i$
        SCHED$\leftarrow$SCHED$\cup\{l_{ij}\}$
        schedule $l_{ij}$'s operation in vector $v$
    for all nonzeros $l_{jk} \in$ SCHED do
        if all operations in row $j$ have been scheduled then
            for all off-diagonal nonzeros $l_{ij}$ in column $j$ do
                enqueue $l_{ij}$ in queue $Q_i$
    $v \leftarrow v + 1$
```

Figure 2: Vectorization algorithm

once, and so each $l_{ij}$ will be enqueued exactly once. Because the algorithm runs until all queues are empty, each $l_{ij}$ will eventually be dequeued and its operation scheduled.

These observations make it easy to prove by induction on the number of off-diagonal nonzeros that each nonzero's operation is scheduled exactly once, so the generated schedule satisfies constraint 2. Finally, the algorithm enqueues only off-diagonal nonzeros from the matrix $L$, so the generated schedule satisfies constraint 1.

## 3.1. Optimality of the vectorization algorithm

Because the running time for a given computation goes down as the average vector length goes up, the optimal vector schedule for solving $Ly = b$ is the schedule with the fewest vectors. The next results establish the optimality of the vectorization algorithm.

**Definition 1.** *An off-diagonal nonzero $l_{ij}$ is enabled at $v$ in a vector schedule if the number of operations from row $j$ in vectors 1 through $v - 1$ equals the number of off-diagonal nonzeros in row $j$.*

Note that if a nonzero $l_{ij}$ is not enabled at $v$ in a given vector schedule, scheduling $l_{ij}$'s operation in vector $v$ violates constraint 4.

**Lemma 1.** *$l_{ij}$ is enabled at $v$ in the vector schedule $G$ generated by the algorithm if and only if the algorithm enqueued $l_{ij}$ before scheduling vector $v$.*

**Proof.** If there are no off-diagonal nonzeros in row $j$, then the algorithm enqueued $l_{ij}$ before scheduling vector 1. If there are off-diagonal nonzeros in row $j$ and $l_{ij}$ is enabled at $v$ in $G$, then vectors 1 through $v - 1$ contain all operations from row $j$. Find the $v' < v$ such that vector $v'$ contains the last operation from row $i$. The algorithm enqueued $l_{ij}$ after scheduling this last operation in vector $v'$.

If the algorithm enqueued $l_{ij}$ before scheduling vector 1, then there are no off-diagonal nonzeros in row $j$ and $l_{ij}$ is enabled in $G$ for all $v$. Otherwise, the algorithm enqueued $l_{ij}$ because it scheduled the last nonzero in row $j$ in some vector $v'$, with $v' < v$. Therefore, the number of operations from row $j$ in vectors 1 through $v'$ equals the number of nonzeros in row $j$, and $l_{ij}$ is enabled at $v$ in $G$. ▮

**Theorem 1.** *Any vector schedule that solves $Ly = b$, where $L$ has unit-diagonals, contains at least as many vectors as the schedule generated by the algorithm.*

**Proof.** Assume there exists a vector schedule $S$ that is shorter than the schedule $G$ generated by the algorithm. Then for some $v$ and row $j$, $S$ has more operations from row $j$ in vectors 1 through $v$ than $G$. Find the smallest such $v$. Then $G$ has no operation from row $j$ in vector $v$, while $S$ does. Therefore, $Q_j$ was empty when the algorithm scheduled vector $v$. Let $t$ equal the number of operations from row $j$ in vectors 1 through $v - 1$ in schedule $G$. Because $Q_j$ was empty when the algorithm scheduled vector $v$, $t$ nonzeros had been enqueued on $Q_j$ when the algorithm scheduled vector $v$. By our lemma, there are $t$ nonzeros from row $j$ in the set of enabled nonzeros at $v$ in $G$.

Because for all rows $i$, $G$ has at least at many operations from row $i$ in vectors 1 through $v - 1$ as $S$ has, the set of enabled nonzeros at $v$ in $G$ is a superset of the set of enabled nonzeros at $v$ in $S$. But, because $S$ has an operation from row $j$ in vector $v$, the number of enabled nonzeros at $v$ in $S$ from row $j$ is at least $t + 1$, which is a contradiction. ▮

This vectorization algorithm runs in time proportional to $n$ plus the number of off-diagonal nonzeros of $L$. The queue initialization phase runs in time proportional to $n$. It is possible to perform the traversal of nonempty queues in time proportional to the number of nonempty queues by maintaining a linked list of nonempty queues. It is also possible to enqueue and dequeue nonzeros in constant time by keeping a pointer to the first and last nonzero in each queue.

Given this information, it is easy to see that the scheduling phase of the algorithm runs in time proportional to the number of enqueue operations plus the number of dequeue operations. Because each off-diagonal nonzero is enqueued once and dequeued once, the scheduling phase of the algorithm takes time proportional to the number of off-diagonal nonzeros.

## 3.2. Removing the Unit Diagonal Assumption

To correctly schedule a matrix with non-unit diagonal elements, the vector schedule must divide each element $y_i$ by $l_{ii}$ after all operations of the form $y_i \leftarrow y_i - l_{ij} \times y_j$.

Once the division, $y_i \leftarrow y_i/l_{ii}$ has been carried out, the algorithm may schedule all operations of the form $y_j \leftarrow y_j - l_{ji} \times y_i$. The algorithm can augment the vector schedule by inserting the divide operation $y_i \leftarrow y_i/l_{ii}$ just before the first vector containing an operation of the form $y_j \leftarrow y_j - l_{ji} \times y_i$. Using this modification, the augmented vector schedule may no longer be optimal.

To maintain optimality of the solve, any lower triangular matrix $L$ with non-unit diagonals may be scaled so as to have unit diagonals. This is accomplished by dividing each column $j$ of $L$ by $l_{jj}$, yielding a new system $\tilde{L}D = L$, where the diagonal matrix $D = \mathrm{diag}(l_{11}, l_{22}, \ldots, l_{nn})$. The work to solve $\tilde{L}Dy = b$ now includes a vectorizable division with $D$ and an optimal solve with $\tilde{L}$.

In practice, most algorithms yield a factorization $B = LU$ with $L$ having unit diagonals and $U$ non-unit diagonals. In this case $U$ may scaled so as to have unit diagonals by dividing each row $i$ of $U$ by $1/u_{ii}$. The resulting system, $B = LDU$, now includes a diagonal matrix $D = \mathrm{diag}(u_{11}, u_{22}, \ldots, u_{nn})$. Using this form of the factorization, the work to solve $By = b$ now includes a solve with $L$, a vectorizable division with the matrix $D$ ($y_i \leftarrow y_i/u_{ii}$, $i = 1, 2, \ldots, n$), and an optimal solve with the newly scaled $U$.

## 4. Computational Results

In this section we compare numerical results for two methods of solving lower triangular systems. Method M1 is the naive method presented in Section 2 and vectorizes the computation associated with each column of the triangular system. Method M2 is an implementation of the vectorization algorithm presented in Section 3. Both methods were incorporated into MINOS/SC 5.3 [Eld89], a modification of the mathematical programming system MINOS 5.3 [MS87]. MINOS/SC includes an alternative basis updating scheme as well as special pricing routines designed especially for vector computers. The computational tests demonstrate the efficiency of the new method and show that M2 is more efficient than method M1 on a representative set of large, sparse problems.

The computational results presented in this section come from timing various portions of MINOS/SC using methods M1 and M2 during the solution of 30 linear programming test problems. Many of these problems are available from the *netlib* collection [Gay85]. The test problem specifications are given in Table 1 in the appendix. The very smallest *netlib* test problems were omitted from the results as the total time required by solutions with the $L_0$ factors for these problems was less than 1/100th of a second on the *Cray Y-MP*.

### 4.1. Results

The computational tests were performed on an 8 processor *Cray Y-MP* supercomputer. The operating system was *UNICOS*, version 5.1, and the MINOS code was compiled using the *CFT77* compiler with full optimization. Each run was made as a batch job.

For each test run the number of iterations and total solution time is recorded in Table 3 in the appendix. The solution time was measured by timing the MINOS subroutine M5SOLV. The options used for MINOS were the standard MINOS/SC vectorizing options, i.e. PARTIAL PRICE 10, SCALE OPTION 2, SCHUR-COMPLEMENT AUTO. The set of problems was then run with and without the VECTORIZE SOLVE option. Note that by "turning off" the VECTORIZE SOLVE option, we do not inhibit the "naive" vectorization of solves with the regular columns of $L_0$. In this case, only the new vectorization algorithm is inhibited. Since the default FACTORIZATION FREQUENCY for MINOS 5.3 is 100, the vectorization algorithm was rerun for the new basis approximately every 100 iterations. This means that each time a new $L_0$ was reordered using the vectorization algorithm, it was used to solve 100 or fewer systems of equations.

For purposes of evaluating the scheduling algorithm, the following items were deemed to be of interest for each method:

1. Total and average time spent solving with $L_0$.
2. Total time spent scheduling for the vectorization algorithm (for VECTORIZE SOLVE option only).
3. Percent increase in vector length.
4. Average vector length in $L_0$ factor.

Time spent in solving with $L_0$ was measured by timing the appropriate portion of the MINOS subroutine LU6SOL and its counterpart for the solve generated by the vectorization algorithm. The total and average solve time with $L_0$ factors, total scheduling time and average vector length, are recorded in Table 2.

The results from Table 2 dramatize how short the average column lengths are for the standard method. The vector lengths for method M1 are given in the column labeled $\mu_{L_0}$. Vector lengths this short will perform poorly on a vector machine. In fact, MINOS/SC has compiler directives to disable vectorization when dealing with vectors shorter than 5. (Without this compiler directive, results for method M1 on the test set were much worse than those given in Table 2.) Under method M2, using the vectorization algorithm, we are able to increase the vector lengths without performing more operations during the solves. Vector lengths for method M2 are given in the column labled $\mu_V$. The average increase in vector length can be seen in the last row of Table 2.

The average solve time is indicated by columns labeled $\bar{S}_L$ for method M1 and $\bar{S}_V$ for method M2. Method M2 gives more efficient solve times for each problem. The speed-up factor for solves with method M2 over that of M1 is given in the column headed $\bar{S}_L/\bar{S}_V$. The aggregate speed-up in solve times for the test set was 3.39.

To determine the success of method M2, the total solve time for method M1 must be compared with the combined solve time and ordering time under method M2. For example, the problem *greenbea* exhibits a 26-fold increase in vector length using the vectorization method. The resulting total speedup for solves with $L_0$ is about 2.3 (i.e. 22.92 seconds for M1 *vs* $(3.79 + 6.06) = 9.85$ seconds for M2). The results of Table 2 show that the total scheduling time for the new algorithm was usually small in comparison with the total solve time with $L_0$. For many of the problems, especially the larger and most difficult ones, the time reduction for the total scheduling time *plus* the total time for solving with $L_0$ under method M2 is only a fraction of the total solve time with $L_0$ under method M1. Allowing more iterations between refactorizations may allow the method to be more competitive, as the scheduling time for method M2 can be amortized over more solves with the factor. For the problems tested, with the exception of *pilots*, the number of solves required to "break even" with the vectorization algorithm ranged from 5.74 to 76.87 with a mean of 21.02.

It is important to note that for every problem tested except *pilots*, the total scheduling time *plus* the solve time with $L_0$ for method M2 was at least as small as the total solve time for the problem run without the vectorization algorithm (M1). This indicates that even for small problems, using the vectorization method does not increase computation time, and for most large problems (the problems of interest for supercomputers) a large reduction in computation time is expected.

The performance on the problem *pilots* is of interest. We note that the overall vector length increase for the scheduling algorithm is not small in relation to many of the other large problems, but as method M1 does create relatively long vectors initially for *pilots*, 11.48 *vs* an average of 2 94 for the test set, increasing the average length using method M2 gives little improvement.

As one might expect, the speedup in solves with $L_0$ under the new method is also related to the percent increase in vector length using method M2. The percent increase in vector length is given in Table 3. The average increase was 895%.

## 4.2. Further work

The vectorization of the solve has only been implemented here for the lower triangular factor $L_0$. The computational results for the solves with $L_0$ have shown that the new scheduling method is efficient for multiple solves with similarly patterned systems. To completely vectorize the solves in the simplex method, the scheduling algorithm should be applied to $L_0$, $L_0^T$, $U_0$, and $U_0^T$ (assuming $U_0$ is not destroyed by the update), each time the current basis is refactorized. This should lead to even more favorable overall timing results. Similarly, in the symmetric case, the vectorization algorithm should be run for $L_0$ and $L_0^T$. This doubles the storage requirement over that in traditional solve methods, but allowing the factor to be stored in two ways makes the forward and back substitution algorithms run more efficiently and can decrease solve times. This is due to the fact that transposed solves using a factor stored by rows (columns) require searching for the column (row) indices needed at each stage in the new vectorized solve. The authors feel that in this day of inexpensive memory, the benefits will be seen on all but the largest of problems where doubling the storage requirement may not be feasible.

The vectorization algorithm should work well for other applications, such as interior-point methods for linear programming. In many of these implementations, a Cholesky factorization $L_k L_k^T = AD_k A^T$ is carried out using the structure of a matrix $AA^T$. The factors are either used directly or as a preconditioner for a conjugate-gradient method to solve a least-squares problem. Although the iteration counts are usually much lower for interior-point methods than for simplex methods, the structure of each system stays constant throughout the algorithm. This means that when the Cholesky factors are used to solve the least-squares problems directly, the scheduling algorithm need be carried out only once for $L_0$ and once for $L_0^T$, and can be used to vectorize the solve for each iteration. In this case the overhead of the scheduling algorithm can be amortized over many solves. When the Cholesky factor is used as a preconditioner, the vectorization algorithm is even more efficient since the factors are used to solve multiple systems each iteration.

## 4.3. Conclusions

When solvir ıltiple large, sparse systems of linear equations it is possible to take advantage of ιne nonzero structure of the triangular factors to increase vector length and decrease computation time when running on a vector computer.

The vectorization algorithm runs in time proportional to the number of nonzero elements and the order of the system. For symmetric systems, and methods that require solves with transposed factors, the scheduling algorithm must be run once for each transposed and untransposed factor.

The gain in efficiency lies in being able to amortize the scheduling cost over multiple solves with the same or identically structured factor.

The vectorization algorithm was incorporated into the mathematical programming system MINOS. Tests on a representative set of large, sparse linear programming test problems showed that on average the length of the vectors arising in the triangular solves increased from 3 to 24, giving an average speedup of 3.4.

## Acknowledgements

The authors are indebted to Michael Saunders for helpful suggestions and to Bill Kamp and John Gregory of Cray Research, Inc. for supporting the research by providing computer time for the computational tests.

## References

[Bar71] Bartels, R.H. (1971). A stabilization of the simplex method. *Numerische Mathen.atik* **16**, 414–434.

[Eld89] Eldersveld, S.K. (1989). MINOS/SC User's Guide Supplement, Internal technical report, Department of Industry, Science and Technology, Cray Research, Inc., Mendota Heights, Minnesota.

[ES90] Eldersveld, S.K. and Saunders, M.A. (1990). A block-*LU* update for large-scale linear programming. SOL technical report (to appear), Department of Operations Research, Stanford University, Stanford, California.

[FT72] Forrest, J.J.H. and Tomlin J.A. (1972). Updating triangular factors of the basis to maintain sparsity in the product form simplex method. *Mathematical Programming* **2**, 263–278.

[Gay85] Gay, D.M. (1985). Electronic mail distribution of linear programming test problems, *Mathematical Programming Society COAL Newsletter*, 13, 10–12.

[GMSW87] Gill, P.E., Murray, W., Saunders, M.A., and Wright, M.H. (1987). Maintaining *LU* factors of a general sparse matrix, *Linear Algebra and its Applications*, 88/89 239–270.

[MS87] Murtagh, B.A. and Saunders, M.A. (1983). MINOS 5.1 User's Guide, Report SOL 83-20R, Department of Operations Research, Stanford University, Stanford, California.

## A. Tables

| No. | Problem | Rows | Cols | Elems | Objective value |
|---|---|---|---|---|---|
| 1 | 80bau3b | 2263 | 2266 | 29063 | 9.8722822814E+05 |
| 2 | bp822 | 822 | 825 | 11127 | 5.5018458595E+03 |
| 3 | cycle | 1904 | 1907 | 21322 | -5.2263930249E+00 |
| 4 | czprob | 930 | 933 | 14173 | 2.1851966988E+06 |
| 5 | etamacro | 401 | 404 | 2489 | -7.5571519542E+02 |
| 6 | fffff800 | 525 | 528 | 6235 | 5.5567961167E+05 |
| 7 | ganges | 1310 | 1313 | 7021 | -1.0958627396E+05 |
| 8 | greenbea | 2393 | 2396 | 31499 | -7.2462397960E+07 |
| 9 | grow22 | 441 | 444 | 8318 | -1.6083433648E+08 |
| 10 | nesm | 663 | 666 | 13988 | 1.4076079892E+07 |
| 11 | perold | 626 | 629 | 6026 | -9.3807558690E+03 |
| 12 | pilot.ja | 941 | 944 | 14706 | -6.1131579663E+03 |
| 13 | pilot.we | 723 | 726 | 9218 | -2.7201045880E+06 |
| 14 | pilot4 | 411 | 414 | 5145 | -2.5811392641E+03 |
| 15 | pilotnov | 976 | 979 | 13129 | -4.4972761882E+03 |
| 16 | pilots | 1442 | 3652 | 43220 | -5.5760732709E+02 |
| 17 | scfxm2 | 661 | 664 | 5229 | 3.6660261565E+04 |
| 18 | scfxm3 | 991 | 994 | 7846 | 5.4901254550E+04 |
| 19 | scrs8 | 491 | 494 | 4029 | 9.0429998619E+02 |
| 20 | scsd6 | 148 | 151 | 5666 | 5.0500000078E+01 |
| 21 | scsd8 | 398 | 401 | 11334 | 9.0499999993E+02 |
| 22 | sctap3 | 1491 | 1494 | 17554 | 1.4240000000E+03 |
| 23 | ship08l | 779 | 782 | 17085 | 1.9090552114E+06 |
| 24 | ship12l | 1152 | 1155 | 21597 | 1.4701879193E+06 |
| 25 | ship12s | 1152 | 1155 | 10941 | 1.4892361344E+06 |
| 26 | stair | 357 | 360 | 3857 | -2.5126695119E+02 |
| 27 | stocfor2 | 2158 | 2161 | 9492 | -3.9024408538E+04 |
| 28 | tdesg1 | 3500 | 4050 | 18041 | 4.3560773922E+04 |
| 29 | tdesg5 | 4215 | 22613 | 105002 | 4.3407357993E+04 |
| 30 | woodw | 1099 | 1102 | 37478 | 1.3044763331E+00 |

Table 1: Problem specifications.

| *Method* | *M1:* | $\bar{S}_L$ | $\mu_{L_0}$ | *M2:* | | $\bar{S}_V$ | $\mu_{V_L}$ | $\bar{S}_L/\bar{S}_V$ |
|---|---|---|---|---|---|---|---|---|
| *Problem name* | *Total lsolve time (sec)* | *Mean lsolve time (μsec)* | *Mean vector length* | *Total schd time (sec)* | *Total lsolve time (sec)* | *Mean lsolve time (μsec)* | *Mean vector length* | *Lsolve speed-up* |
| 80bau3b | 6.21 | 5500.55 | 1.58 | 0.59 | 0.96 | 869.93 | 39.74 | 6.38 |
| bp822 | 2.31 | 4228.79 | 3.51 | 0.37 | 0.87 | 1580.87 | 26.46 | 2.67 |
| cycle | 1.63 | 5673.91 | 3.16 | 0.25 | 0.45 | 1499.64 | 51.18 | 3.78 |
| czprob | 0.33 | 2407.53 | 1.00 | 0.02 | 0.08 | 603.62 | 6.66 | 3.99 |
| etamacro | 0.07 | 1150.58 | 1.45 | 0.01 | 0.02 | 282.38 | 13.83 | 4.07 |
| fffff800 | 0.05 | 762.92 | 3.96 | 0.01 | 0.03 | 487.24 | 11.79 | 1.57 |
| ganges | 0.10 | 1443.91 | 1.66 | 0.01 | 0.03 | 371.01 | 16.04 | 3.89 |
| greenbea | 22.92 | 13870.9 | 3.46 | 3.79 | 6.06 | 3513.42 | 90.24 | 3.95 |
| grow22 | 0.30 | 3234.55 | 5.45 | 0.07 | 0.21 | 1985.22 | 21.29 | 1.63 |
| nesm | 0.23 | 1012.18 | 1.46 | 0.03 | 0.08 | 311.88 | 12.15 | 3.25 |
| perold | 1.14 | 3847.43 | 4.16 | 0.23 | 0.50 | 1669.09 | 25.58 | 2.30 |
| pilot.ja | 2.44 | 4728.59 | 4.55 | 0.48 | 1.07 | 2106.41 | 30.52 | 2.25 |
| pilot.we | 1.59 | 4566.55 | 2.92 | 0.20 | 0.40 | 1200.97 | 36.73 | 3.81 |
| pilot4 | 0.26 | 2223.11 | 5.47 | 0.06 | 0.19 | 1662.52 | 16.80 | 1.34 |
| pilotnov | 0.84 | 4225.29 | 3.93 | 0.15 | 0.36 | 1809.30 | 25.31 | 2.34 |
| pilots | 11.70 | 8687.68 | 11.48 | 4.87 | 11.73 | 8484.91 | 47.09 | 1.02 |
| scfxm2 | 0.11 | 1584.88 | 2.13 | 0.01 | 0.03 | 455.72 | 19.72 | 3.48 |
| scfxm3 | 0.29 | 2546.77 | 2.22 | 0.04 | 0.07 | 613.37 | 30.11 | 4.15 |
| scrs8 | 0.07 | 1291.28 | 1.48 | 0.01 | 0.02 | 326.17 | 13.84 | 3.96 |
| scsd6 | 0.09 | 961.20 | 1.88 | 0.01 | 0.03 | 312.56 | 13.34 | 3.08 |
| scsd8 | 0.69 | 2644.14 | 1.87 | 0.06 | 0.16 | 576.46 | 23.76 | 4.59 |
| sctap3 | 0.05 | 543.33 | 1.53 | 0.01 | 0.02 | 186.59 | 10.39 | 2.91 |
| ship08l | 0.10 | 1756.20 | 1.06 | 0.01 | 0.03 | 444.42 | 7.73 | 3.95 |
| ship12l | 0.19 | 1858.74 | 1.02 | 0.01 | 0.03 | 311.47 | 13.44 | 5.97 |
| ship12s | 0.05 | 971.57 | 1.01 | 0.01 | <0.01 | 147.06 | 23.29 | 6.61 |
| stair | 0.08 | 1644.06 | 5.72 | 0.02 | 0.06 | 1197.79 | 17.64 | 1.37 |
| stocfor2 | 0.44 | 2215.02 | 1.54 | 0.05 | 0.08 | 419.07 | 29.00 | 5.29 |
| tdesg1 | 2.82 | 7377.14 | 2.11 | 0.31 | 1.14 | 2974.36 | 11.00 | 2.48 |
| tdesg5 | 30.48 | 13914.05 | 2.19 | 3.33 | 10.39 | 4474.36 | 15.15 | 3.11 |
| woodw | 0.90 | 3380.22 | 3.14 | 0.12 | 0.35 | 1295.92 | 20.59 | 2.61 |
| *MEAN* | 2.95 | 3675.10 | 2.94 | 0.50 | 1.18 | 1405.79 | 24.01 | 3.39 |

Table 2: Problem Results.

| *Method* | *M1:* | | | *M2:* | | | |
|---|---|---|---|---|---|---|---|
| *Problem name* | *Itns* | *Soln time (secs)* | *Time Per itn (millisecs)* | *Itns* | *Soln time (secs)* | *Time Per itn (millisecs)* | *% incr in vect len* |
| 80bau3b | 11750 | 155.24 | 13.21 | 11571 | 150.96 | 13.05 | 2310.9 |
| bp822 | 6346 | 51.89 | 8.18 | 6346 | 51.01 | 8.04 | 668.6 |
| cycle | 3302 | 40.82 | 12.36 | 3359 | 40.61 | 12.09 | 1517.7 |
| czprob | 1434 | 8.38 | 5.84 | 1434 | 8.17 | 5.70 | 566.4 |
| etamacro | 726 | 2.70 | 3.72 | 719 | 2.64 | 3.67 | 876.7 |
| fffff800 | 754 | 3.61 | 4.79 | 772 | 3.71 | 4.81 | 189.2 |
| ganges | 711 | 4.94 | 6.95 | 711 | 4.84 | 6.81 | 742.0 |
| greenbea | 19194 | 351.14 | 18.29 | 20070 | 353.7 | 17.62 | 2527.6 |
| grow22 | 1072 | 7.01 | 6.54 | 1201 | 7.86 | 6.54 | 342.7 |
| nesm | 2549 | 13.08 | 5.13 | 2851 | 14.74 | 5.17 | 714.8 |
| perold | 3655 | 24.10 | 6.59 | 3655 | 23.72 | 6.49 | 546.8 |
| pilot.ja | 6265 | 54.30 | 8.67 | 6182 | 52.91 | 8.56 | 588.5 |
| pilot.we | 4202 | 30.60 | 7.28 | 3962 | 27.87 | 7.03 | 1182.4 |
| pilot4 | 1352 | 6.18 | 4.57 | 1354 | 6.21 | 4.59 | 233.5 |
| pilotnov | 2340 | 21.88 | 9.35 | 2340 | 21.58 | 9.22 | 604.7 |
| pilots | 15549 | 326.56 | 21.00 | 15966 | 341.03 | 21.36 | 329.3 |
| scfxm2 | 755 | 3.92 | 5.19 | 835 | 4.17 | 4.99 | 798.4 |
| scfxm3 | 1281 | 9.25 | 7.22 | 1281 | 9.07 | 7.08 | 1242.6 |
| scrs8 | 669 | 2.60 | 3.89 | 669 | 2.55 | 3.81 | 858.8 |
| scsd6 | 1040 | 3.07 | 2.95 | 1228 | 3.46 | 2.82 | 663.3 |
| scsd8 | 2706 | 15.20 | 5.62 | 2864 | 15.54 | 5.43 | 1276.9 |
| sctap3 | 1008 | 7.51 | 7.45 | 1008 | 7.49 | 7.43 | 537.8 |
| ship08l | 591 | 3.11 | 5.26 | 591 | 3.04 | 5.14 | 639.3 |
| ship12l | 1012 | 6.95 | 6.87 | 1012 | 6.81 | 6.73 | 1217.5 |
| ship12s | 538 | 3.31 | 6.15 | 538 | 3.26 | 6.06 | 2205.9 |
| stair | 609 | 2.68 | 4.40 | 609 | 2.69 | 4.42 | 231.2 |
| stocfor2 | 2196 | 27.03 | 12.31 | 2111 | 25.48 | 12.07 | 1812.4 |
| tdesg1 | 4099 | 73.55 | 17.94 | 4099 | 72.77 | 17.75 | 401.6 |
| tdesg5 | 22935 | 619.03 | 26.99 | 24350 | 643.04 | 26.41 | 584.8 |
| woodw | 2747 | 27.99 | 10.19 | 2831 | 28.21 | 9.96 | 476.4 |
| *MEAN* | 4113 | 63.59 | 8.83 | 4217 | 63.09 | 8.69 | 895.7 |

Table 3: Overall Problem Results.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

| REPORT DOCUMENTATION PAGE | | READ INSTRUCTIONS BEFORE COMPLETING FORM |
|---|---|---|
| 1. REPORT NUMBER<br>SOL 90-1 | 2. GOVT ACCESSION NO. | 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE (and Subtitle)<br>A Vectorization Algorithm for the Solution of Large, Sparse Triangular Systems of Equations | | 5. TYPE OF REPORT & PERIOD COVERED<br>Technical Report |
| | | 6. PERFORMING ORG. REPORT NUMBER |
| 7. AUTHOR(s)<br>Samuel K. Eldersveld and Martin C. Rinard | | 8. CONTRACT OR GRANT NUMBER(s)<br>N00014-90-J-1242 |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS<br>Department of Operations Research - SOL<br>Stanford University<br>Stanford, CA 94305-4022 | | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS<br>1111MA |
| 11. CONTROLLING OFFICE NAME AND ADDRESS | | 12. REPORT DATE<br>January 1990 |
| | | 13. NUMBER OF PAGES<br>12 pp. |
| | | 15. SECURITY CLASS. (of this report)<br>UNCLASSIFIED |
| | | 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE |

16. DISTRIBUTION STATEMENT (of this Report)

This document has been approved for public release and sale; its distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

[illegible]

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

triangular systems; linear programming; vector computers . [illegible]

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Please see other side...

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

Technical Report SOL 90-1*
January 1990

## Abstract

A new method is given for use with vector computers on applications that require multiple solutions with identically patterned triangular factors and different right-hand sides. A key feature is that a vectorization algorithm is used to place the nonzeros from the factors in a few long vectors. The method is shown to work well when incorporated into the mathematical programming system MINOS and tested on 30 linear programming test problems. [illegible]

[illegible]